图文 050、动手实验：使用 jstat 摸清线上系统的JVM运行状况

详情 评论

动手实验：

使用jstat摸清线上系统的JVM运行状况

狸猫技术窝专栏上新，基于**真实订单系统**的消息中间件（mq）实战，重磅推荐：

未来3个月，我的好朋友原子弹大侠将带你一起，全程实战，360度死磕MQ

（点击下方蓝字进行试听）

从 0 开始带你成为消息中间件实战高手

重要说明：

如何提问：每篇文章都有评论区，大家可以尽情在评论区留言提问，我都会逐一答疑

（**ps**：评论区还精选了一些小伙伴对**专栏每日思考题的作答**，有的答案真的非常好！大家可以通过看别人的思路，启发一下自己，从而加深理解）

如何加群：购买了狸猫技术窝专栏的小伙伴都可以加入**狸猫技术交流群**。

（群里有不少**一二线互联网大厂的助教**，大家可以一起讨论交流各种技术）

具体**加群方式**请参见文末。

（**注**：以前通过其他专栏加过群的同学就不要重复加了）

1、前文回顾

上周我们已经通过带着大家分析GC日志的方式，给大家重新回顾了一遍JVM的整体运行原理，包括对象优先在Eden区分配，Young GC的触发时机和执行过程，对象进入老年代的时机，Full GC的触发时机和执行过程，相信大家通过GC日志可以把JVM整体运行原理理解的更加的深入而且透彻。

本周我们就要带着大家开始用一个工具来分析运行中的系统，他的对象增长的速率，Young GC的触发频率，Young GC的耗时，每次Young GC后有多少对象是存活下来的，每次Young GC过后有多少对象进入了老年代，老年代对象增长的速率，Full GC的触发频率，Full GC的耗时。

2、功能强大的jstat

平时我们对运行中的系统，如果要检查他的JVM的整体运行情况，比较实用的工具之一，就是jstat

他可以轻易的让你看到当前运行中的系统，他的JVM内的Eden、Survivor、老年代的内存使用情况，还有Young GC和Full gC的执行次数以及耗时。

通过这些指标，我们可以轻松的分析出当前系统的运行情况，判断当前系统的内存使用压力以及GC压力，还有就是内存分配是否合理。下面我们就一点点来看看这个jstat工具的使用。

3、jstat -gc PID

首先第一个命令，就是在你们的生产机器linux上，找出你们的Java进程的PID，这个大家自行百度一下即可，用jps命令就可以看到。

接着就针对我们的Java进程执行：jstat -gc PID。这就可以看到这个Java进程（其实本质就是一个JVM）的内存和GC情况了。

运行这个命令之后会看到如下列，给大家解释一下：

S0C：这是From Survivor区的大小
S1C：这是To Survivor区的大小
S0U：这是From Survivor区当前使用的内存大小
S1U：这是To Survivor区当前使用的内存大小
EC：这是Eden区的大小
EU：这是Eden区当前使用的内存大小
OC：这是老年代的大小
OU：这是老年代当前使用的内存大小
MC：这是方法区（永久代、元数据区）的大小

MU：这是方法区（永久代、元数据区）的当前使用的内存大小
YGC：这是系统运行迄今为止的Young GC次数
YGCT：这是Young GC的耗时
FGC：这是系统运行迄今为止的Full GC次数
FGCT：这是Full GC的耗时
GCT：这是所有GC的总耗时

不知道大家发现什么没有，其实这些指标都是非常实用的jvm gc分析指标，接下来我们一步一步告诉大家该怎么使用这个工具。

另外给大家说句题外话，接下来我们有两篇文章会给大家一段模拟出生产案例的程序，然后在windows本地电脑上运行，然后我们会带着大家用jstat工具去分析他的jvm运行情况。

但是jstat工具本身如果要在windows上运行需要使用专门针对windows的版本，所以后面的文章我们会教会大家怎么在windows上使用jstat工具的。

4、其他的jstat命令

除了上面的jstat -gc命令是最常用的以外，他还有一些命令可以看到更多详细的信息，如下所示：

jstat -gccapacity PID：堆内存分析
jstat -gcnew PID：年轻代GC分析，这里的TT和MTT可以看到对象在年轻代存活的年龄和存活的最大年龄
jstat -gcnewcapacity PID：年轻代内存分析
jstat -gcold PID：老年代GC分析
jstat -gcoldcapacity PID：老年代内存分析
jstat -gcmetacapacity PID：元数据区内存分析

大家可以后面自己有机会尝试尝试这些命令，多看看，还是挺好玩儿的，但是其实最完整、最常用、最实用的还是jstat -gc命令，基本足够我们日常分析jvm的运行情况了。

5、到底该如何使用jstat工具?

接着教教大家一些jstat工具使用的小技巧，先明确一下，我们分析线上的JVM进程，最想要知道的信息有哪些?

包括如下：新生代对象增长的速率，Young GC的触发频率，Young GC的耗时，每次Young GC后有多少对象是存活下来的，每次Young GC过后有多少对象进入了老年代，老年代对象增长的速率，Full GC的触发频率，Full GC的耗时。

只要知道了这些信息，其实我们就可以结合之前几周的文章分析过的JVMGC优化的方法，合理分配内存空间，尽可能让对象留在年轻代不进入老年代，避免发生频繁的Full GC。这就是对JVM最好的性能优化了!

因此我们一点点分析，通过jstat工具如何得到上述信息。

6、新生代对象增长的速率

如果认真看过之前几周文章的同学，应该都知道，我们平时对jvm第一个要了解的事儿，就是随着系统运行，每秒钟会在年轻代的Eden区分配多少对象。

要分析这东西，你只要在线上linux机器上运行如下命令：jstat -gc PID 1000 10

这行命令，他的意思就是每隔1秒钟更新出来最新的一行jstat统计信息，一共执行10次jstat统计

通过这个命令，你可以非常灵活的对线上机器通过固定频率输出统计信息，观察每隔一段时间的jvm中的Eden区对象占用变化。

比如给大家举个例子，执行这个命令之后，第一秒先显示出来Eden区使用了200MB内存，第二秒显示出来的那行统计信息里，发信Eden区使用了205MB内存，第三秒显示出来的那行统计信息里，发现Eden区使用了209MB内存，以此类推。

此时你可以轻易的推断出来，这个系统大概每秒钟会新增5MB左右的对象。

而且这里大家可以根据自己系统的情况灵活多变的使用，比如你们系统负载很低，不一定每秒都有请求，那么可以把上面的1秒钟调整为1分钟，甚至10分钟，去看你们系统每隔1分钟或者10分钟大概增长多少对象。

还有就是一般系统都有高峰和日常两种状态，比如系统高峰期用的人很多，此时你就应该在系统高峰期去用上述命令看看高峰期的对象增长速率。然后你再得在非高峰的日常时间段内看看对象的增长速率。

按照上述思路，基本上你可以对线上系统的高峰和日常两个时间段内的对象增长速率有很清晰的了解。

7、Young GC的触发频率和每次耗时

接着下一步我们就想知道大概多久会触发一次Young GC，以及每次Young GC的耗时了。

其实多久触发一次Young GC就很容易推测出来了，因为系统高峰和日常时候的对象增长速率你都知道了，那么非常简单就可以推测出来高峰期多久发生一次Young GC，日常期多久发生一次Young GC。

比如你Eden区有800MB内存，那么发现高峰期每秒新增5MB对象，大概高峰期就是3分钟会触发一次Young GC。日常期每秒新增0.5MB对象，那么日常期大概需要半个小时才会触发一次Young GC。

那么每次Young GC的平均耗时呢?

简单，之前给大家说过，jstat会告诉你迄今为止系统已经发生了多少次Young GC以及这些Young GC的总耗时。

比如系统运行24小时后共发生了260次Young GC，总耗时为20s。那么平均下来每次Young GC大概就耗时几十毫秒的时间。

你大概就知道每次Young GC的时候会导致系统停顿几十毫秒。

8、每次Young GC后有多少对象是存活和进入老年代

接着我们想要知道，每次Young GC后有多少对象会存活下来，以及有多少对象会进入老年代。

其实每次Young GC过后有多少对象会存活下来，这个没法直接看出来，但是有办法可以大致推测出来。

之前我们已经推算出来高峰期的时候多久发生一次Young GC，比如3分钟会有一次Young GC

那么此时我们可以执行下述jstat命令：jstat -gc PID 180000 10。这就相当于是让他每隔三分钟执行一次统计，连续执行10次。

此时大家可以观察一下，每隔三分钟之后发生了一次Young GC，此时Eden、Survivor、老年代的对象变化。

正常来说，Eden区肯定会在几乎放满之后重新变得里面对象很少，比如800MB的空间就使用了几十MB。Survivor区肯定会放入一些存活对象，老年代可能会增长一些对象占用。所以这里的关键，就是观察老年代的对象增长速率。

从一个正常的角度来看，老年代的对象是不太可能不停的快速增长的，因为普通的系统其实没那么多长期存活的对象。如果你发现比如每次Young GC过后，老年代对象都要增长几十MB，那很有可能就是你一次Young GC过后存活对象太多了。

存活对象太多，可能导致放入Survivor区域之后触发了动态年龄判定规则进入老年代，也可能是Survivor区域放不下了，所以大部分存活对象进入老年代。

最常见的就是这种情况。如果你的老年代每次在Young GC过后就新增几百KB，或者几MB的对象，这个还算情有可缘，但是如果老年代对象快速增长，那一定是不正常的。

所以通过上述观察策略，你就可以知道每次Young GC过后多少对象是存活的，实际上Survivor区域里的和进入老年代的对象，都是存活的。

你也可以知道老年代对象的增长速率，比如每隔3分钟一次Young GC，每次会有50MB对象进入老年代，这就是年代对象的增长速率，每隔3分钟增长50MB。

9、Full GC的触发时机和耗时

只要知道了老年代对象的增长速率，那么Full GC的触发时机就很清晰了，比如老年代总共有800MB的内存，每隔3分钟新增50MB对象，那么大概每小时就会触发一次Full GC。

然后可以看到jstat打印出来的系统运行起劲为止的Full GC次数以及总耗时，比如一共执行了10次Full GC，共耗时30s，每次Full GC大概就是需要耗费3s左右。

10、本文总结

通过本文对jstat命令的介绍，以及结合我们之前学习过的jvm运行原理，我们教给了大家这套分析线上系统jvm运行情况的技巧

大家完全可以灵活运行jstat这个实用的工具，轻而易举的掌控到线上jvm运行的详细情况，然后针对jvm的具体运行情况去进行有针对性的优化。

另外，很多同学会问了：老师，其实有很多其他的工具也特别好用啊，比如JConsole、VisualVM等可视化的监控工具，还有其他一些开源的监控系统，都是可视化的。

针对这个问题，其实我也没说不可以用那些可视化工具，下篇文章我们就要给大家介绍更多的可视化监控JVM的工具。

但是有一点要告诉大家，一个优秀、合格的工程师，他一定是可以非常灵活的运用各种命令行工具，在命令行就搞定一切的。

所以jstat作为一个最简单易用、高效实用的命令行jvm监控工具，其实绝对是值得大家首先掌握他的。因为每个人的公司情况不一样，万一你公司不支持你用各种可视化工具呢？那你就必须从最“low”最原始的命令行工具开始，快速上手实用，定位问题。

而且其实你理解了本文的思想之后，你用其他任何工具，都能轻松的把线上jvm的运行情况通过工具提供的数据分析清楚。

11、今日思考题

今天交给大家一个练习题，就是在自己线上负责的系统使用jstat命令，按照上述我们介绍的思路，把以下jvm运行情况全部摸出来：

新生代对象增长的速率
Young GC的触发频率
Young GC的耗时
每次Young GC后有多少对象是存活下来的
每次Young GC过后有多少对象进入了老年代
老年代对象增长的速率
Full GC的触发频率
Full GC的耗时

如果有分析心得的，可以发评论区里，跟其他同学一起分享，如果在落地过程中遇到任何问题，也欢迎大家在评论区踊跃发言提问。

End
